REMARKS/ARGUMENTS

The amendments made herein are in response to the office action mailed April 07, 2004 (Office Action). This response is being filed with a petition for a one month retroactive extension of time with the appropriate fee.

The amendment filed January 22, 2004 was objected to under 35 U.S.C. § 132 as introducing new matter into the disclosure. In particular, the amendment that stated that Listener object stub remotely calls a method that is executed in a third address space was objected to as the Examiner interpreted the specification as showing "query for the stub reference from a third process address space and store in the first address space and when the event occurs, utilizing the stub reference to call a method in the second address space" as indicated in paragraph 6 of the Office Action.

In response, Applicants amended claims 1, 4, 9, and 12 (as they existed before the last attempt to amend, to which the examiner objected) to show that the Notifier and Listener are located in separate programming spaces and that a routine that permits the Notifier to call a method in the Listener is located in yet another programming space (a third programming space).

Support for this amendment can be found on page 15, lines 7-21 as noted by the Examiner in paragraph 6 of the Office Action. Further support can be found in FIG. 3 that shows three different Java Virtual Machines (JVMs) 8, 9, and 10; the Notifier 13 being located in JVM 10, the Listener 12 being located in JVM 8, and the method invoking routine (RMI Naming Registry 11) being located in JVM 9. No new matter has been added responsive to this amendment.

{WP192955:1}

Responsive to the amendment above, the 35 U.S.C. § 132 objection of paragraph 4 and the 35 U.S.C. § 112 objection of paragraphs 5-6 are no longer applicable. Applicants respectfully request that these objections be withdrawn.

Applicants note that in asserting the previous objections, the Examiner interpreted the claims as "such that said Listener object stub remotely calls a method that is executed in the second process addressing space" for examining purposes. Such an interpretation rendered the amendment and subsequent argument without meaning, thereby allowing the Examiner to re-assert formerly presented arguments.

Claims 1-2, 4-5, 9-10, and 12-13 stand rejected under 35 U.S.C. § 103(a), as being unpatentable over a Riehle, Dirk [1996] "The Event Notification Pattern: Integrating Implicit Invocation with Object-Orientation" in Theory and Practice of Object Systems, 2, 1, pages 43-52 (Riehle). Claims 3, 6-8, 11, and 14-16 stand rejected under 35 U.S.C. § 103(a), as being unpatentable over Riehle in view of OMG TC Document 95.8.19 [1995] "COM/CORBA Interworking RFP Part A" (OMG) and in further view of "Sun Microsystems, Inc. Remote Method Invocation Specification" (Sun RMI).

Prior to addressing the rejections on the art, a brief review of the Applicant's invention is in order. Applicants claim a method of placing a Notifier in a first address space, a listener in a second address space, and an invoker (the RMI Naming Registry 11 is the invoker) in a third address space. This arrangement permits the Notifier to invoke methods in one or more of the Listeners located in a different address space from the Notifier responsive to the events in the first address space. The invocation occurs transparently due to the invoker in the third address

space. In one embodiment, each of the three referenced address spaces can be located within a different Java Virtual Machine (JVM).

Referring to claims 1, 4, 9, and 12 Applicants claim:

* a **Notifier** and a **Listener stub** both located **in a first address space**
* a **Listener** associated with the listener stub located in a **second address space**
* a **routine in a third address space**, the routine being invoked by the Notifier responsive to an event in the first address space. Once invoked, the routine calls a designated method of the Listener, thus causing the method to execute in the second address space.

Reihle, as noted by the Examiner in paragraph 10, fails to teach that the Notifier is to be disposed in a first address space, that the Listener is to be in the second address space. Further, Reihle fails to teach that the invocation is to occur through an invoker in a third address space. The Examiner asserts, however that placing the Listener, Notifier, and Invoker in different address spaces would have been obvious. As proof, the Examiner references page 8 of Riehle stating that the IACEvent link serves to make notification transparent to process boundaries. The cited reference, however, **cannot** function in the matter suggested by the Examiner.

Riehle shows that the Event link is an abstract class and that EventStub and IACEvLink inherit the forward method from the Event link class. Inheritance **requires** that both the classes to reside within **the same address space**. For example, a class which inherits methods from another class in the JAVA programming language **must** reside within the same virtual machine as the abstract class from which it inherits a method. Consequently the proposition (asserted as obvious) is **not possible** based upon the structure shown in FIG. 4 of Riehle. Accordingly, Riehle fails to teach or suggest the Applicants invention as claimed.

Further (referring specifically to claims 4 and 12), the Examiner claims that location transparent event handling is taught by Riehle in particular at page 3, section 2.2. The
{WP192955;1}

referenced section states " This pattern lets developers decouple dependent objects from those objects they depend on, lets them make the abstract state and dependencies of objects explicit in class interfaces, ... " This fails to teach **location transparent** event handling. The teachings of "decoupling" means coupling the stated feature from a particular object or software routine. That is, an abstract state can be formed, as shown in FIG. 4. This abstract state, however, does nothing to make **location** transparent.

In fact, the location shown in FIG. 4 (the **IACAddress) must be explicitly** referenced by the IACEvLink object. The IACEvLink object (because it inherits a method from Event Link) **must** be in the **same processing space** as **the Event Link object**. Likewise, as shown, the Observer referenced by the EventStub (that fails to include a mechanism for including an address) **must be** in the same address space as the Observer, as shown in FIG. 4.

Accordingly, Reihle fails to teach or suggest location transparent event handling, as explicitly claimed by the Applicants – "wherein said **Notifier instance** and said **Listener instance** are configured to perform **location transparent event handling.**"

In light of the above, Reihle fails to teach or suggest the Applicants claimed invention. Accordingly, the 35 U.S.C. § 103(a) rejections to claims 1-2, 4-5, 9-10, and 12-13 should be withdrawn, which action is respectfully requested.

Referring to paragraphs 19-24, claims 3, 6-8, 11, and 14-16 have been rejected as unpatentable over Riehle, in view of OMG in further view of Sun RMI. OMG and Sun RMI fail to cure the deficiencies of Riehle as neither individually or in combination teach or suggest:

* a **Notifier** and a **Listener stub** both located **in a first address space**
* a **Listener** associated with the listener stub located in a **second address space**

> * **a routine in a third address space**, the routine being invoked by the Notifier responsive to an event in the first address space. Once invoked, the routine calls a designated method of the Listener, thus causing the method to execute in the second address space.

which are claimed limitations of the Applicants' invention.

Further, there is no motivation to combine the teachings of Riehle with the teachings of OMG and Sun RMI. Absence the Applicants disclosure, no motivation to combine the references in the manner suggested by the Examiner exists. Moreover, one of ordinary skill in the art cannot combine the cited references without contradicting the explicit teachings of the references.

Specifically, teachings in OMG should not be combined with Riehle since OMG teaches away from Riehle. The event handling disclosed by OMG at page 4, paragraph 2 states that events can be managed by mapping OLE Custom Controls (OCX's) to propagate events via OCX interfaces. That is, OLE objects in CORBA require a coupling between communication objects and an interface. Further, according to the CORBA specification, only CORBA objects with interfaces can be accessed remotely. Riehle, however, discloses at page 1, paragraph 5, that interdependent software objects should not be coupled to interfaces or interface objects so that system evolution and maintenance is facilitated. Consequently, the teachings of Riehle and OMG conflict.

Additionally, the Examiner assumes that CORBA and SUN RMI teachings are interchangeable and combinable. This viewpoint is incorrect for many reasons. RMI and CORBA are different, competing technologies for establishing communications between distributed objects. A variety of structural and functional differences exist between the RMI and

CORBA, resulting in the technologies being incompatible with one another. Examples of some significant deviations between RMI and CORBA include:

- RMI interfaces are defined in Java, while CORBA interfaces are defined in Interface Definition Language (IDL).
- RMI relies heavily upon Java Object Serialization to function and can be run upon any platform that implements a JVM. In contrast, CORBA relies upon ORB libraries to be written that adhere to the CORBA specification to function and are not natively operable within a JVM.
- RMI passes local objects by copying them and passes remote objects by reference allowing new classes to be passed across JVM for execution (mobile code). In contrast, CORBA supports passing parameters according to input and output definitions allowing only primitive data types and structures to be passed, as opposed to actual code.
- RMI automatically performs garbage collecting on distributed objects using mechanisms located within a JVM, while CORBA objects garbage collection is not performed upon CORBA objects.
- RMI is a model designed for a single language where all objects are written in Java. CORBA is a language independent specification.

The RMI and CORBA technologies have different advantages and disadvantages over one another. Event handling concepts that function within CORBA cannot be simply integrated into the competing RMI model as suggested by the Examiner. In fact, porting CORBA event handling concepts so that such functions could operate within an RMI model would require substantial innovations not known within the relevant art at the time of the invention. Applicants solicit the Examiner to show teaching of such heretofore unknown innovations that would permit the CORBA and RMI references to be combined in the manner suggested.

For reasons above, Reihle, OMG, Sun RMI, and combinations thereof fail to teach or suggest the Applicants claimed invention. Consequently, the 35 U.S.C. § 103(a) rejections to claims 3, 6-8, 11, and 14-16 should be withdrawn, which action is respectfully requested.

{WP192955;1}

Applicants believe that this application is now in full condition for allowance, which action is respectfully requested. The Applicants request that the Examiner call the undersigned if clarification is needed on any matter within this Amendment, or if the Examiner believes a telephone interview would expedite the prosecution of the subject application to completion.

Respectfully submitted,

Date: 8/9/04

Gregory A. Nelson, Registration No. 30,577
Richard A. Hinson, Registration No. 47,652
Brian K. Buchheit, Registration No. 52,667
AKERMAN SENTERFITT
Post Office Box 3188
West Palm Beach, FL 33402-3188
Telephone: (561) 653-5000

{WP192955;1}